Evolução do Sistema AIDA Centralizado para um Ambiente Concorrente usando OMT e SDL-92

Carla Geovana N. Macário[a], Moacir Pedroso Jr.[b] e Walter C. Borelli[c]

[ab] Centro Nacional de Pesquisa Tecnológica em Informática para a Agricultura - CNPTIA/EMBRAPA. CP 6041, 13083-970, Campinas SP, Brasil.
(carla, pedroso)@cnptia.embrapa.br

[c] Departamento de Telemática – Faculdade de Engenharia Elétrica e de Computação, UNICAMP, CP 6101, 13081-970, Campinas SP, Brasil.
borelli@dt.fee.unicamp.br

Resumo

Este trabalho propõe o uso combinado da metodologia OMT com a linguagem de especificação formal SDL para a evolução do AIDA, um software para o gerenciamento e análise de dados experimentais, em desenvolvimento na EMBRAPA, partindo de uma versão centralizada para um ambiente concorrente. Esta abordagem apresenta facilidades que produzem ganhos no processo de desenvolvimento de software, como a possibilidade de validação e de simulação do sistema, e também a geração de código para sua prototipação. São apresentados os modelos de objetos e as especificações em SDL das duas versões, o resultado da validação do ambiente AIDA centralizado e também um exemplo de simulação deste sistema.

Palavras-chave: Specification and Description Language - SDL, Object Modeling Technique - OMT, especificação formal, validação, simulação, reuso.

Evolução do Sistema AIDA Centralizado para um Ambiente Concorrente usando OMT e SDL-92

1. Introdução

O Ambiente Integrado para Desenvolvimento e Análise - AIDA consiste numa evolução do Software NTIA - SW NTIA [7], um ambiente de software que começou a ser desenvolvido no Centro Nacional de Pesquisa Tecnológica em Informática para a Agricultura - CNPTIA/EMBRAPA, em 1986 para suprir uma carência que existia na empresa: a inexistência de produtos voltados para a análise matemática e estatística de dados que pudessem ser executados em microcomputadores.

As pesquisas agropecuárias geram um grande volume de dados a serem analisados utilizando-se modelos matemáticos e estatísticos. Até 1986, todas as análise de dados da empresa eram realizadas num computador de grande porte central, localizado na unidade central da EMBRAPA. Assim, para a realização das análises necessárias, os pesquisadores das unidades enviavam seus dados em papel para a sede da empresa, onde a análise era executada. Entretanto, tal procedimento apresentava muitas dificuldades, principalmente a ocorrência de erros nos resultados, decorrentes de preenchimento errado dos formulários que continham os dados, ou mesmo na sua digitação para o processamento no computador.

Diante disto, identificou-se a necessidade de desenvolvimento de um software para a análise matemática e estatística de dados, que pudesse ser executado em microcomputadores, viabilizando a realização das análises pelos próprios pesquisadores nas unidades descentralizadas. Assim, teve início o desenvolvimento do software NTIA, que foi evoluído até 1995, incorporando um conjunto de novos módulos segundo as necessidades dos seus usuários, e passando a ser executado também em ambientes UNIX.

Entretanto, a atividade de manutenção deste software era difícil, devendo-se principalmente à metodologia utilizada no seu desenvolvimento, que não permitia a incorporação fácil de novas funcionalidades. Isto dificultava bastante o atendimento rápido das solicitações dos usuários, bem como a incorporação de novas tecnologias, como por exemplo a adoção de uma interface gráfica baseada em janelas e um sistema de armazenamento de dados mais eficiente do que o utilizado.

Foi então proposto, em meados de 1995, o desenvolvimento do AIDA[2], um novo ambiente de software voltado para a análise, apresentando as vantagens do SW NTIA, mas com uma linguagem de programação única, e utilizando algumas das tecnologias disponíveis, como Sistema Gerenciador de Banco de Dados - SGBD e interface gráfica. A idéia era aproveitar todo o conhecimento acumulado no desenvolvimento do SW NTIA, reestruturando-o de forma a incorporar novos conceitos que facilitassem o desenvolvimento de novos módulos e, principalmente, a utilização de tecnologias emergentes.

O AIDA é um ambiente de software multiplataforma voltado para o domínio de análise de dados, provido de interface simples, clara e eficiente, que permite a fácil incorporação de novas funcionalidades, viabiliza a incorporação de tecnologias e ferramentas emergentes, e integra facilmente novos métodos de análise de dados. As ferramentas que realizam a análise dos dados são organizadas em grupos: estatísticas descritivas, estatísticas básicas, freqüências e correlações; modelos lineares: análise de variância uni e multivariada, regressão linear e stepwise; pesquisa operacional: programação linear usando o algoritmo simplex revisado; gráficas: gráficos de barra, coluna, setorial, 2D, 3D e curvas de níveis.

A execução de uma ferramenta gera um relatório padrão, que pode ser exibido na tela ou enviado para a impressora. Em alguns casos também são gerados arquivos de saída. Cada sessão de trabalho do AIDA armazena todas as execuções realizadas, permitindo que elas sejam recuperadas num outro momento. Assim, torna-se possível manter um histórico das análises realizadas, que podem ser acessadas sempre que necessário. Os relatórios gerados pelas

ferramentas podem ser editados através do Editor de Relatórios, que oferece comandos para a personalização dos relatórios produzidos. Arquivos escritos em um formato diferente do AIDA, podem ser utilizados no ambiente após serem convertidos pelo Conversor de Formatos.

Buscando-se sistematizar o desenvolvimento do AIDA e prover um entendimento comum do sistema pelos desenvolvedores, bem como facilitar futuras evoluções, foi adotada a Object Modeling Technique - OMT [11], por ser uma metodologia orientada a objetos mais completa, que aborda diferentes aspectos do sistema em questão. Neste sentido, foram elaborados os modelos propostos pela OMT para o AIDA[3], especificando-se uma arquitetura básica do sistema que permitisse a incorporação de novas funcionalidades e facilidades.

Assim, a partir dos modelos propostos, foi apresentada, em meados de 1996 no CNPTIA, uma versão de demonstração do sistema AIDA centralizado para o ambiente MS-WINDOWS, que contemplava apenas ferramentas estatísticas. Nesta versão centralizada, apenas uma única pessoa pode executar o sistema, executando somente uma ferramenta por vez. Basicamente, esta versão consistiu numa reestruturação do SW NTIA, com a aplicação de conceitos de objetos.

Nesta experiência inicial foram identificadas algumas dificuldades. Em primeiro lugar a falta de uma ferramenta que facilitasse a utilização da metodologia. Em segundo lugar, e principalmente, a impossibilidade de simulação e de validação do sistema ainda nas fases de especificação e de desenho (*design*), o que permitiria a identificação de erros e garantiria um custo menor no desenvolvimento do sistema como um todo.

Neste sentido, apesar da OMT se mostrar uma excelente opção para a descrição de software, não consiste numa técnica formal de especificação, não permitindo a geração de código a partir da especificação para a simulação ou a validação do sistema em desenvolvimento, ou mesmo a sua prototipação. Desta forma, os ganhos provenientes com estas atividades não são possíveis apenas com o uso desta metodologia.

Este trabalho propõe a utilização da metodologia combinada[8] de OMT com a linguagem de especificação formal SDL - Specification and Description Language [5] na especificação da evolução do sistema AIDA de um ambiente centralizado para um concorrente, apresentando os ganhos obtidos no processo de desenvolvimento de software. Os modelos em OMT e SDL aqui apresentados representam sistemas preparados para evoluírem do sistema centralizado para o concorrente, ou deste para um outro, consistindo em generalizações das versões centralizada e concorrente propostas em [9] e [10].

Para a automação desta metodologia, utilizou-se o SDT[1] - SDL Design Tool, um ambiente composto por várias ferramentas para especificação, simulação e validação de sistemas em SDL.

2. Metodologia Proposta

Inicialmente é feita uma descrição informal do sistema, buscando-se obter o máximo de detalhes possível. Partindo-se desta descrição, passa-se ao modelo de objetos proposto pela OMT, onde é feita a identificação das classes que irão compor o software, com a descrição do escopo de cada uma, seus requisitos e suas restrições. Em seguida devem ser identificadas as associações (relações) entre as classes, e os seus atributos. Durante todo este processo deve-se explorar os conceitos de reuso e de herança de classes, procurando-se elaborar um modelo enxuto e conciso.

Deste modelo de objetos, passa-se à elaboração da especificação do sistema em SDL. Como a versão mais recente do SDL, o SDL-92[5], incorpora conceitos de objetos do tipo

[1]SDT Pacote SDT, versão 3.02 (SDT Base, MSC Editor, Simulator e Validator): adquirido pelo DT/FEEC/UNICAMP através de Projeto Temático - FAPESP (Proc. 91/3660-0).

instancias, classes, reuso e herança, o mapeamento do modelo de objetos para a linguagem SDL torna-se uma atividade simples e quase automática. Além disso, a possibilidade de especificar o sistema com diferentes níveis de abstração, permite a sua utilização nas diversas fases do processo de desenvolvimento, partindo da especificação até a sua implementação.

Uma especificação em SDL apresenta uma identificação geral do sistema, que é feita através de blocos, canais e tipos abstratos de dados. A especificação dos blocos é feita através de processos e sinais. Já nos processos é descrito realmente o comportamento do sistema, em termos de estados e transições. Neste nível os sinais são apresentados já com os seus parâmetros, caso os mesmos existam. Dependendo do nível de abstração utilizado, pode-se chegar até ao detalhamento da manipulação de dados.

3. Modelo de Objetos do AIDA Centralizado

O modelo de objetos apresentado na Figura 1 consiste numa versão mais geral do que aquela proposta em [9], onde não se previa a sua evolução para um ambiente concorrente. Este modelo é o resultado da análise do software, representando a estrutura geral do AIDA, num alto nível de abstração. Na verdade, cada uma das classes apresentadas poderiam ser descritas em outros diagramas. Por exemplo, as classes *EInterf*, *TInterf* e *FCInterf* são especializações de uma classe genérica *Interface*, não representada neste diagrama, da qual herdam características comuns.

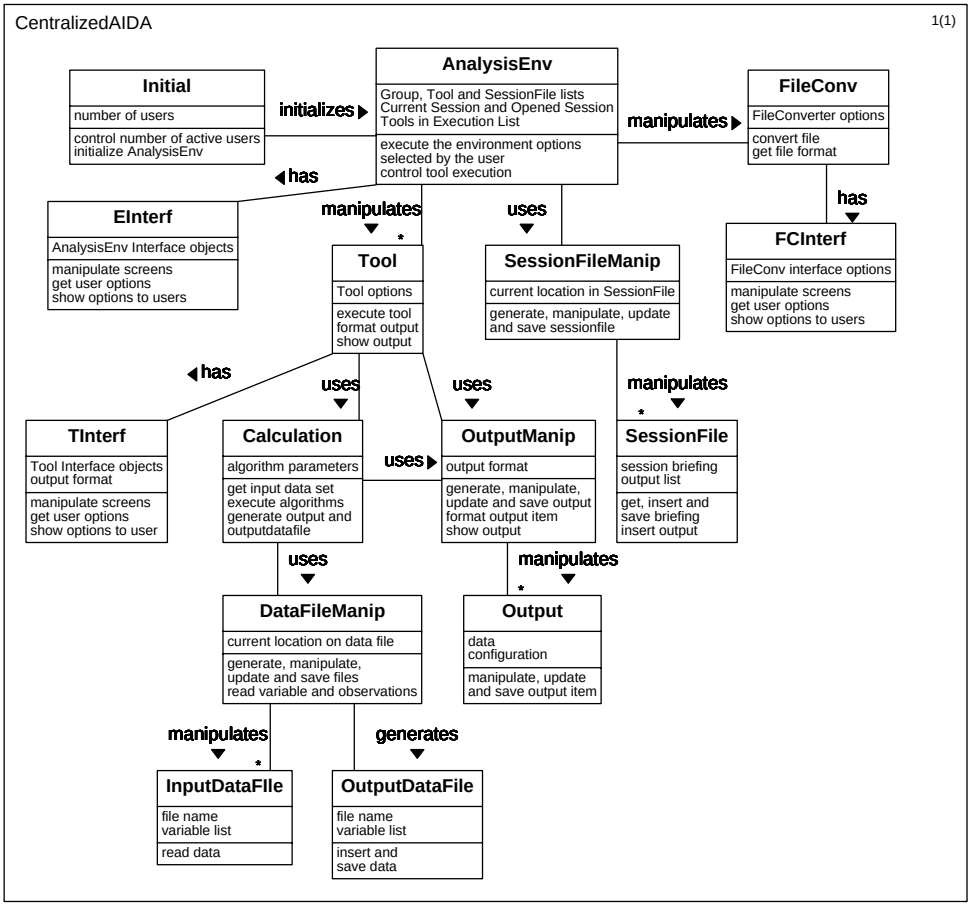

Figura 1- Modelo de Classes do AIDA Centralizado

Durante a análise do sistema, foram identificadas as classes principais do AIDA que representam as suas funcionalidades: ferramentas para análise matemática e estatística, formatador de relatórios e conversor de arquivos.

A classe *Initial*, que não existia na primeira versão do sistema centralizado [9], inicializa o ambiente AIDA e controla o número de usuários executando o sistema. Esta classe é responsável por instanciar a classe *AnalysisEnv* para cada usuário que iniciar a execução do AIDA. Na versão centralizada, esta classe permite o uso do sistema por apenas por uma pessoa por vez.

A classe *AnalysisEnv* (Figura 1), que possui relacionamentos com outras quatro classes: *EInterf*, *Tool*, *SessionFileManip* e *FileConv*, é responsável por gerenciar a execução das ferramentas, bem como do formatador de relatórios. Além disso, também controla as sessões de trabalho do usuário, e as opções selecionadas por ele. A interação entre o usuário e o software AIDA é feita através da classe *EInterf* , que gerencia todos os elementos da interface do ambiente. A classe *SessionFileManip* possui as funções específicas para a manipulação das sessões, e a classe *FileConv* converte arquivos externos para o formato do ambiente AIDA, interagindo com o usuário através da classe *FCInterf*.

A classe *Tool* (Figura 1) representa uma especificação genérica para todas ferramentas incorporadas ao ambiente AIDA. Esta classe é responsável tanto pela execução da ferramenta em si, como pela formatação de um relatório que foi gerado por ela, e usa três outras classes: *TInterf*, que representa a interação entre o usuário e a ferramenta; *OutputManip*, para a formatação de um relatório; e *Calculation*, que executa algoritmos específicos à ferramenta, acessando os arquivos de dados, se necessário. A classe *Calculation* também se relaciona com a classe *OutputManip* na medida em que gera relatórios ao final da execução dos algoritmos. A classe *DataFileManip* manipula e gera arquivos de dados de entrada e saída, respectivamente.

4. Modelo SDL do AIDA Centralizado

A linguagem formal SDL consiste numa padronização da ITU-T, antiga CCITT, usada originalmente para a especificação de sistemas da área de Telecomunicações. Como características principais, tem-se a possibilidade de representar a estrutura do sistema e o seu comportamento, em diferentes níveis de abstração, e a existência de uma linguagem gráfica, denominada SDL-GR, que facilita tanto a sua utilização, como o seu entendimento[1].

Em SDL existem duas formas de declarações: tipos e instâncias. A diferença entre elas é que os tipos podem ser reusados e as instâncias não, consistindo em definições diretas dos elementos. Até a versão 88, os sistemas, os blocos, os processos e os serviços só podiam ser definidos como instâncias, o que resultava em especificações repetidas quando mais de uma instância de um elemento era necessária. No SDL-92 é possível defini-los também como tipos (*system*, *block*, *process* e *service types*), o que permite o seu reuso ou a sua especialização[4].

O modelo em SDL da versão centralizada foi elaborado através de um mapeamento direto do modelo de objetos, de uma maneira quase automática, onde cada objeto dinâmico identificado foi representado em SDL por um *process type*. Este mapeamento é apresentado na Tabela 1.

O *system type CentralizedAIDA* representa o modelo de objetos da Figura 1. Os blocos *AnalysisEnv*, *Tool* e *FileConv* representam os relacionamentos existentes entre suas respectivas classes, e os processos existentes em cada um destes blocos representam a sua funcionalidade.

Como neste processo de desenvolvimento tinha-se em mente a evolução do sistema AIDA centralizado para um ambiente concorrente, bem como a geração de elementos reusáveis para este ou para outros sistemas, optou-se por utilizar estruturas *system type*, *block type* e *process type* para representar as classes identificadas para o AIDA (Figura 1).

Também foram usadas outras funcionalidades presentes na linguagem SDL-92, como o mecanismo de `PACKAGE`, que permite reusar integralmente um sistema já especificado, ou partes dele. Neste sentido, é necessário apenas a especialização de alguns dos processos existentes, através da sua redefinição, e, em alguns casos, a inclusão de outros. Para que um

block type ou um *process type* possa ser redefinido, o mesmo deve ser declarado como `VIRTUAL` na sua especificação original (*virtual block type* ou *virtual process type*). Em especial nos processos, as transições a serem redefinidas também devem ser definidas originalmente como `VIRTUAL`. Assim, como o sistema centralizado foi reusado para a especificação do sistema concorrente, muitos dos diagramas apresentados nesta sessão apresentam este tipo de construção. Apenas o bloco *FileConverter* não foi definido como `VIRTUAL`, pois na evolução do sistema, ele se manteve inalterado, sem ter sido necessária a sua redefinição.

Classes OMT	Representação em SDL
AIDA	System Type CentralizedAIDA, composto pelos block types AnalysisEnv, FileConv e Tool
AnalysisEnv	Block Type AnalysisEnv, composto pelos process types Initial, AnalysisEnv e EInterf
FileConv	Block type FileConv, composto pelos process types FileConv e FCInterf
Tool	Block type Tool, composto pelos process types Tool, TInterf e Calculation
Output, SessionFile, DataFile	Classes passivas a serem representadas como Tipos Abstratos de Dados - TAD
OutputManip, SessionFileManip, DataFileManip	Classes a serem representadas como operações sobre os TAD

Tabela 1 - Mapeamento de OMT para SDL do sistema AIDA Centralizado

O sistema AIDA centralizado (Figura 2), é composto por três blocos, *AnalysisEnv* (Figura 3), *Tool* (Figura 4) e *FileConv* (Figura 5), representados por figuras com bordas duplas, e cujas instâncias (figuras com borda simples) apresentam os respectivos nomes: *AE*, *T* e *FC*. Os *block types* especificam as características de cada classe, e o relacionamento entre elas é representado pelas instâncias, onde são indicadas as trocas de mensagens.

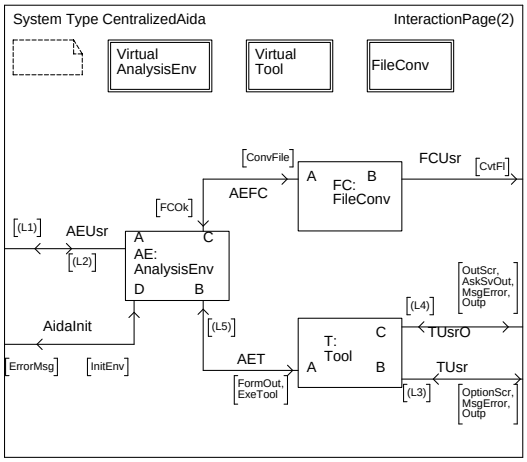

Figura 2- System Type CentralizedAida

Na especificação de blocos, devem ser definidos o número inicial e o número máximo de instâncias que cada processo poderá apresentar simultaneamente durante uma execução do sistema. Estes números são indicados após o nome de cada instância de cada processo que compõe o bloco. Por exemplo, a declaração *Init(1,1)* na especificação do bloco *AnalysisEnv* (Figura 3), indica que existirá no início da execução do sistema uma instância *Init* do processo

Initial, não sendo permitida a criação de nenhuma outra instância deste processo durante esta execução.

O bloco *AnalysisEnv* (Figura 3) é composto por três processos: *Initial*, *AnalysisEnv* e *EInterf*. O processo *AnalysisEnv* é responsável pelo gerenciamento da execução das ferramentas e do conversor de formatos, além de gerenciar as sessões de trabalho. Além disso, cria uma instância do processo *EInterf*, que representa a sua interface, usando para isso o símbolo `CREATE REQUEST` com a informação do nome da instância a ser criada: *EI*. O processo *Initial*, que não existia na primeira especificação do sistema centralizado[9], consiste numa das evoluções decorrentes da validação do sistema. Este processo inicializa o ambiente AIDA e controla o número de usuários executando o sistema, criando uma instância do processo *AnalysisEnv* , denominada *AE*, para cada um deles. Na versão centralizada, este processo só deixa o sistema ser executado por uma pessoa por vez e portanto, apenas uma instância *AE* poderá ser criada.

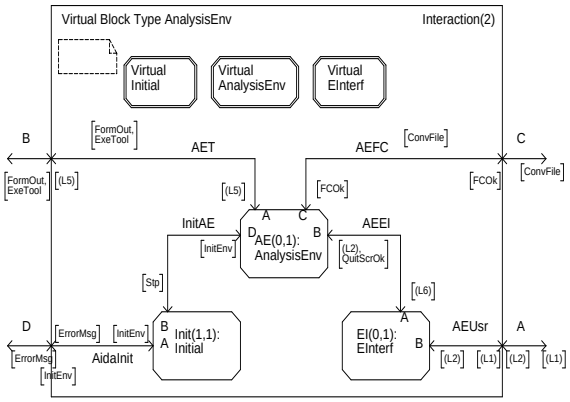

Figura 3 - Virtual Block Type AnalysisEnv

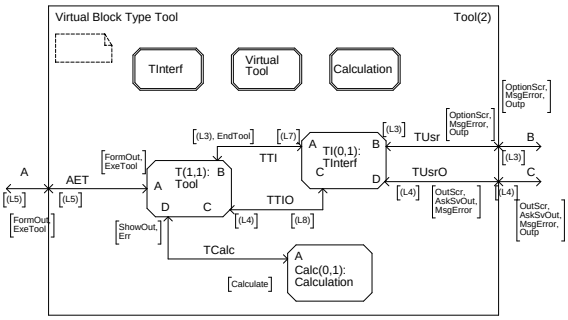

Figura 4 - Virtual Block Type Tool

O bloco *Tool* (Figura 4) é composto pelos processos *Tool*, *TInterf* e *Calculation*, que apresentam as mesmas funcionalidades descritas no modelo de objetos (Figura 1). A classe *Tool* é responsável pela execução da análise, bem como pela formatação de relatório, e *TInterf* pelas interfaces correspondentes. A classe *Calculation* realiza o cálculo dos algoritmos.

O bloco *FileConv* (Figura 5) contém dois processos: *FileConv* e *FCInterf*. O processo *FileConv* é responsável pela conversão de arquivos externos para o formato do AIDA e pela instanciação da interface desta ferramenta, representada pelo process type *FCInterf*.

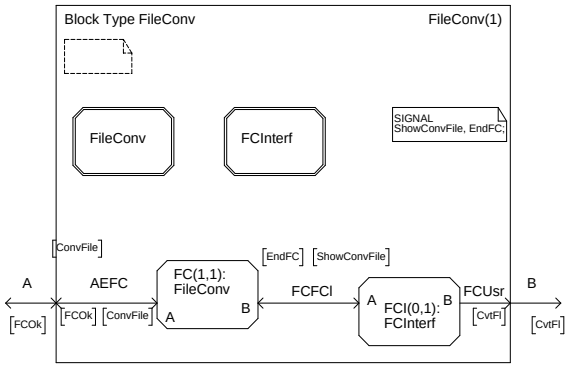

Figura 5 - Block Type FileConv

5. Modelo de Objetos do AIDA Concorrente

Diferente de [10], o modelo de objetos do AIDA concorrente (Figura 6) consiste numa evolução do modelo centralizado (Figura 1), para um ambiente onde o armazenamento de dados e/ou o processamento das ferramentas pode ser feito por servidores diferentes, como por

exemplo em uma rede de PC´s ou de estações de trabalho, de maneira transparente ao usuário. A principal diferença entre o modelo centralizado é a inclusão das classes *Controller* e *Server.*

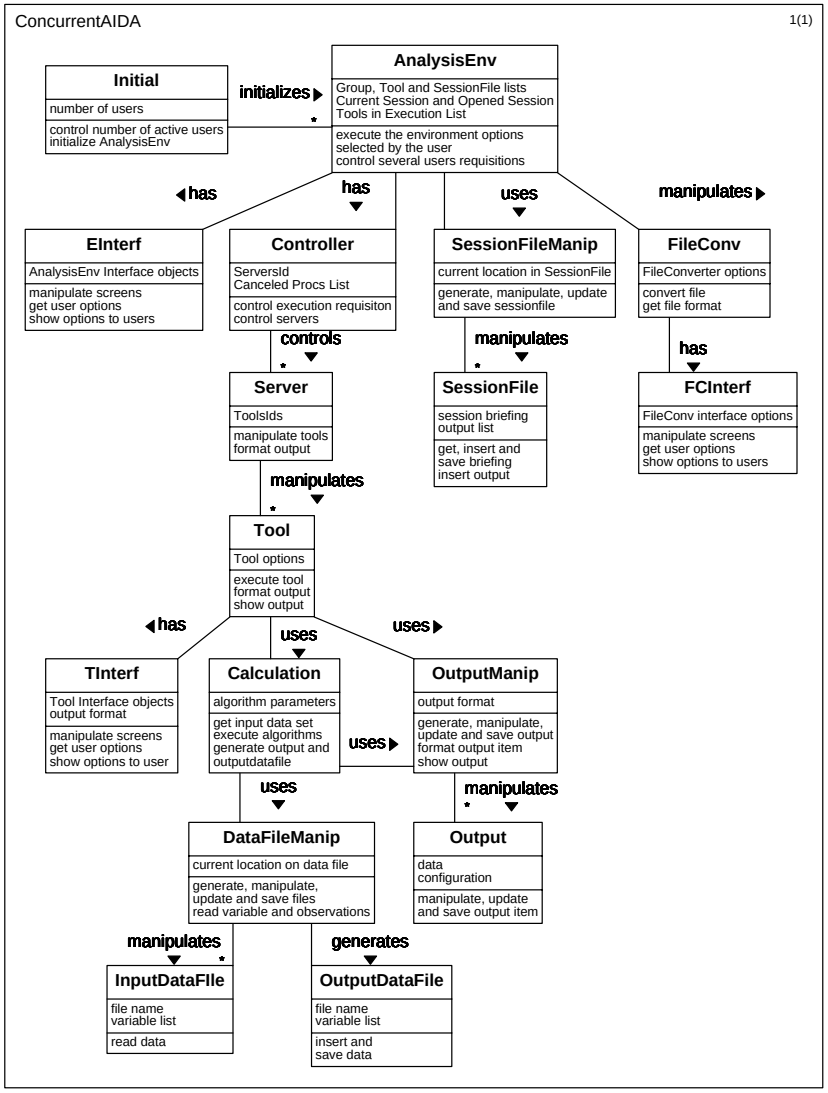

Figura 6- Modelo de Objetos do AIDA Concorrente

A classe *Controller* é responsável por controlar as requisições dos usuários para a execução de ferramentas ou formatação de relatórios. Ao receber uma requisição, esta classe verifica se existe alguma instância de servidor de processamento disponível, e havendo disponibilidade, envia a requisição para o servidor correspondente. Caso contrário, as requisições não atendidas são armazenadas numa fila até que algum dos servidores seja liberado.

A classe *Server* representa o servidor que recebe as requisições do controlador e as envia à classe *Tool.* Na evolução do sistema centralizado para o sistema concorrente, a classe *AnalysisEnv* passa a se relacionar com a classe *Controller*, em lugar da classe *Tool*.

6. Modelo SDL do AIDA Concorrente

Na versão centralizada (Figura 2), existe apenas um usuário que pode executar localmente uma única ferramenta, seja para cálculo ou para formatar um relatório. Já na sua evolução para um ambiente concorrente, vários usuários podem executar o sistema de maneira concorrente, submetendo requisições de execução de ferramentas ou de formatação de relatórios, via *Controller*, às diversas instâncias do processo *Server*. Cada *Server*, por sua vez,

pode executar, ao mesmo tempo, um número determinado de ferramentas, utilizando uma política de processamento do tipo “time-sharing”.

Na elaboração do modelo SDL para a versão concorrente (Figura 7), o sistema centralizado foi reutilizado na sua totalidade através do mecanismo de `PACKAGE` e do uso da construção do tipo `REDEFINED`, que indica a redefinição de um block type ou de um process type que existia na especificação original. Assim, o sistema AIDA concorrente herdou todo o sistema centralizado, redefinindo os blocos *AnalysisEnv* e *Tool* para a inclusão de novos sinais. O bloco *FileConv* permanece como na versão centralizada (Figura 5).

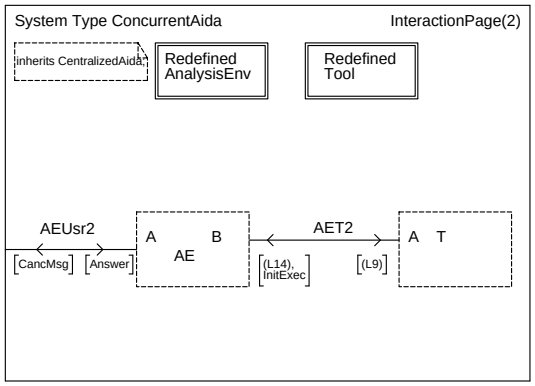

Figura 7- System Type ConcurrentAIDA

Quando um sistema herda outro, devem ser especificadas apenas as diferenças existentes, com o reuso das instâncias definidas anteriormente ou com a definição de novas. Na definição do system type *ConcurrentAIDA* (Figura 7), a construção `INHERITS` indica que o sistema *CentralizedAIDA* está sendo reutilizado e apenas os sinais que não existiam na especificação anterior são definidos. Foram criados novos sinais para controlar quem solicita as requisições e quais as ferramentas que estão sendo executadas. Na versão centralizada isso não era necessário por existir apenas um usuário executando uma única uma ferramenta por vez. Como na versão concorrente são vários usuários executando mais de uma ferramenta ao mesmo tempo, tornou-se necessária a existência de sinais indicando que uma ferramenta teve seu processamento iniciado, ou que foi cancelada, e também quem requisitou esta execução. Na linguagem SDL cada instância de um processo possui uma identificação única, do tipo Pid. Assim, para que o controle de ferramentas e usuários seja correto, os novos sinais possuem parâmetros deste tipo, carregando as identificações correspondentes. Como exemplo na Figura 7 tem-se os sinais *InitExec(Pid)*, que indica que a ferramenta identificada por *Pid* teve seu processamento iniciado, e *CancelExec(Pid, Integer)*, que indica o cancelamento da execução da ferramenta identificada por *Pid*. O parâmetro *Integer* indica o seu índice na tabela que o processo *AnalysisEnv* mantém sobre as execuções de ferramentas que ele solicitou.

Foram definidos novos canais (*AEUsr2* e *AET2*) para transporte dos novos sinais, pois canais não podem ser redefinidos (*AEUsr* e *AET* - Figura 2). Os elementos pontilhados são aqueles que estão sendo reutilizados da especificação anterior, indicando que suas características iniciais serão mantidas, ou seja, serão mantidos os seus processos, suas rotas de sinais e seus sinais. No entanto, novos processos ou sinais podem ser adicionados, como acontece nos block types *AnalysisEnv* e *Tool* (Figura 8 e Figura 9).

Na redefinição do block type *AnalysisEnv* (Figura 3 e Figura 8), os processos *AnalysisEnv* (Figura 10) e *EInterf* passaram a apresentar novas instâncias que receberão os novos sinais, além dos originais. A princípio poderiam ter sido reusadas as instâncias *AE* e *EI* da especificação anterior, mas era necessário mudar o limite máximo de usuários do sistema, que na versão centralizada é igual a um. Diferentemente de [10], foram criadas novas instâncias *AE2(0,)* e *EI2(0,)*, cujo limite de usuários é indeterminado, já que o número máximo de instâncias não é especificado, ficando a cargo do novo processo *Initial* definir este número.

Como *AE2* e *EI2* não são instâncias reusadas, é necessária a declaração de todos os sinais que são recebidos ou enviados por elas, inclusive aqueles que já existiam, como por exemplo *ExeTool* e *(L2)* na Figura 8. Já o processo *Initial* foi redefinido para permitir a execução do sistema por mais de uma pessoa.

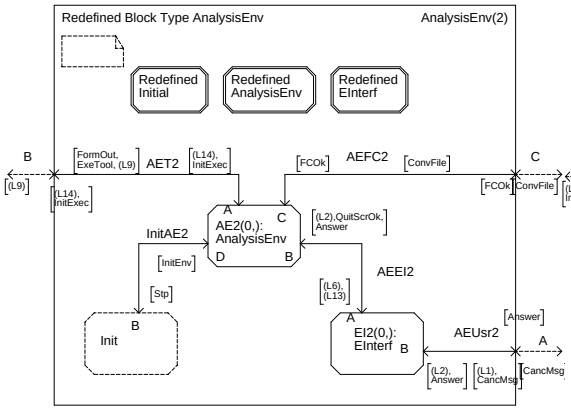

Figura 8 - Redefined Block Type AnalysisEnv

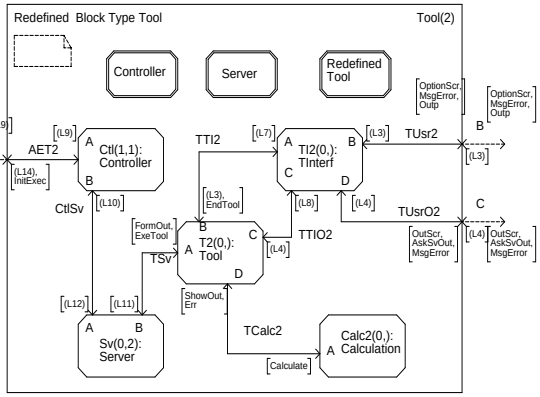

Figura 9 - Redefined Block Type Tool

O block type *Tool*, (Figura 4 e Figura 9), foi redefinido para incluir o novo processo *Controller*, responsável por gerenciar as requisições de execução de ferramentas feitas pelos usuários, e o também novo processo *Server*, que se encarrega de enviar estas requisições para o processo *Tool*, o qual efetivamente executa a ferramenta. Além disso foi definida uma nova instância do processo *Tool*, denominada *T2*, para definir novos números inicial e máximo de instâncias. Assim, diferente da especificação da versão centralizada, no início do sistema não existe nenhuma instância deste processo e deixa de haver limites com relação ao número de ferramentas em execução ao mesmo tempo.

Uma especificação de processos em SDL representa a troca de mensagens e o seu comportamento. Aqui as tarefas foram abstraídas e representadas através do símbolo `TASK`, com um texto descrevendo a ação correspondente. Por exemplo: ‘update session file’ (Figura 10) e ‘verify canceled procs list’ (Figura 11). Os processos *AnalysisEnv*, *Controller* e *Server* são apresentados parcialmente em Figura 10, Figura 11 e Figura 12 respectivamente.

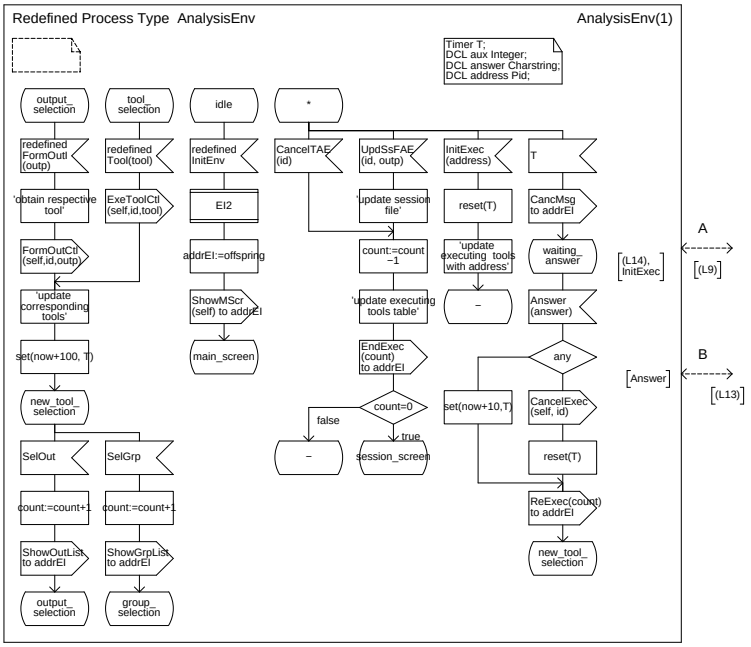

Figura 10 - Redefined ProcessAnalysisEnv (Parcial)

O processo *AnalysisEnv* no sistema concorrente passa a controlar as diversas requisições de execução das ferramentas vindas dos diferentes usuários, informando-os sobre os seus atendimentos. Na versão centralizada este processo enviava um sinal para execução da ferramenta e nada era feito até que chegasse algum resultado da execução. Na versão concorrente, passa a existir um temporizador *T*, que é ativado toda vez que um dos usuários fizer uma requisição de execução. Caso o tempo máximo de espera por uma resposta seja atingido, este processo envia um sinal para processo de interface (*EInterf*), solicitando ao usuário manter ou excluir a sua requisição da fila de processos a serem executados.

O processo *Controller* (Figura 11), no recebimento de uma requisição de execução de ferramenta, verifica a disponibilidade de servidores para atendê-la. Havendo alguma instância de *Server* disponível, envia, então, a requisição de execução para a instância correspondente, esperando o recebimento do sinal *InitExec*, que indica o início do processamento daquela ferramenta. Este sinal, com o Pid da ferramenta em execução, é repassado ao processo *AnalysisEnv* do usuário requisitante, indicando que o seu pedido foi atendido. As eventuais requisições que não podem ser atendidas são colocadas numa fila FIFO para controle de execução de ferramentas, utilizando para isso o símbolo `SAVE` (Figura 11 - estado *waiting_server*, sinais *ExeToolCtl* e *FormOutCtl*), que salva o sinal recebido sem consumi-lo.

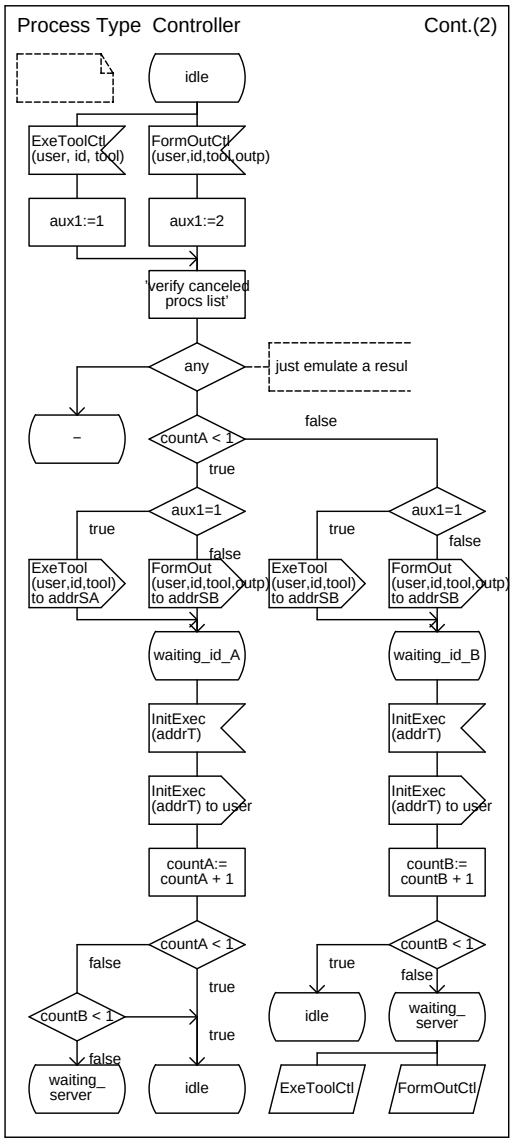

Figura 11- Process Type Controller (Partial)

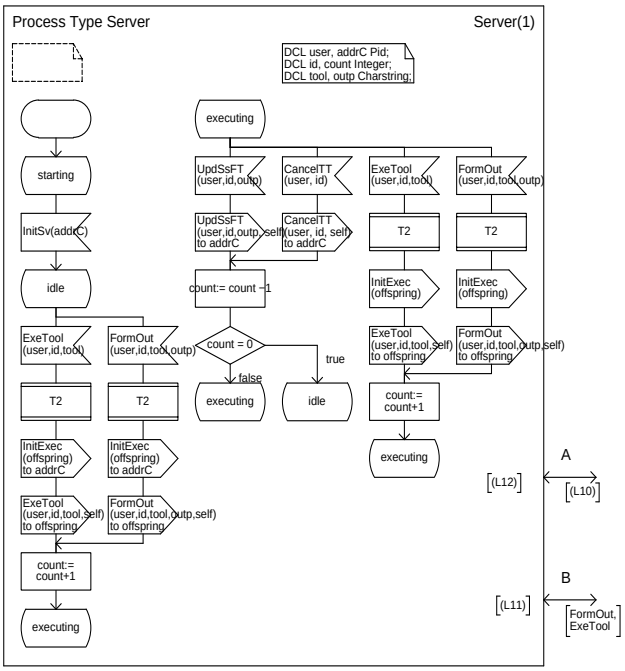

Figura 12- Process Type Server

Assim que um servidor é liberado e está pronto para executar uma nova requisição, o controlador obtém a primeira solicitação da fila e a envia ao processo *Server* (Figura 12). Este processo, ao receber uma requisição de execução, cria uma instância *T2* do processo *Tool*, inicializa a sua execução, retorna ao controlador o sinal *InitExec* com o Pid da ferramenta inicializada e incrementa o contador de ferramentas (*count*) de 1. Na recepção de um sinal que indica o fim de execução de uma ferramenta (sinais *UpdSsFlT* ou *CancelTT*), o servidor decrementa o contador e repassa este sinal para o controlador, indicando que está liberado para atender outra requisição.

O processo *Tool*, que faz parte do bloco *Tool* (Figura 9), foi redefinido para criar as novas instâncias *TI2* e *Calc2* (*TI* e *Calc* na versão centralizada - Figura 4) dos process types *TInterf* e *Calculation*, e para incluir novos sinais com informações adicionais que passaram a ser necessárias.

7. Resultados Obtidos

7.1. Validação do sistema

A validação do sistema permitiu a identificação de falhas na especificação que provavelmente só se tornariam visíveis na fase de teste. Além disso, tornou a correção destas falhas uma atividade simples, já que o SDT[12] possibilita a identificação exata da situação em que elas ocorrem. Os resultados da validação do sistema AIDA concorrente são mostrados na Figura 13.

```
Random-walk

** Starting random walk **
Depth       : 100
Repetitions : 100

** Random walk statistics **
No of reports: 0
Gen states   : 14300
Max depth    : 100
Min depth    : 100
Symbol coverage :  86.96
```

Figura 13 - Resultados da Validação do AIDA Concorrente

Durante o processo de validação, o sistema SDL é representado por uma árvore de comportamento, onde cada nó corresponde a um estado completo do sistema. Cada um destes estados é determinado, entre outras coisas, pelas instâncias ativas de processos que mudaram desde o estado anterior, pelo estado em que elas se encontram e pela suas filas de sinais de entrada. Assim, como o AIDA é um sistema onde vários usuários podem executar um número indeterminado de ferramentas, diferentes instâncias de usuários e de ferramentas são criadas durante a validação, o que leva a um número grande de estados gerados. Em casos como este, é mais apropriado o uso do método *random walk*[12], que realiza uma exploração na árvore de comportamento escolhendo repetidamente caminhos aleatórios para teste, os quais serão realizados segundo regras pré estabelecidas.

Na validação representada pela Figura 13, a árvore de comportamento criada possui uma profundidade máxima (*depth*) de 100, e foram realizadas 100 escolhas (*repetitions*) de caminhos para teste. Segundo os resultados apresentados, não foi encontrada nenhuma situação de erro (*No. of reports*) na validação dos 14.300 estados gerados (*gen. states*), e durante este processo, foram executados 86,96% dos símbolos que compõem o sistema (*symbol coverage*). O resultado diferente de 100% se deve ao fato de que o sistema concorrente herdou o sistema centralizado, redefinindo alguns de seus processos originais. Assim, os símbolos que fazem parte das transições originais deixam de ser executados e são substituídos por aqueles definidos nas redefinições destas transições.

Esta validação é o resultado de outras tentativas de validação em que o número de situações de erros (*reports*) foi diferente de zero, como por exemplo a existência de sinais que chegavam para processos que não se encontravam em estados preparados para recebê-los.

Com a identificação destas falhas na especificação é possível retornar aos processos em SDL para sua correção. Em alguns casos, estas correções podem levar a alterações também no modelo de objetos, como aconteceu no AIDA. Através da sua validação, tornou-se clara a necessidade da existência do processo *Initial* no modelo SDL para inicializar o sistema AIDA e controlar o número de usuários (Figura 3 e Figura 8), levando também à revisão do modelo OMT, onde foi incluída a classe *Initial* (Figura 1).

7.2. Exemplo de Simulação

A simulação do sistema através do *Simulator*[12], consiste numa outra forma de validar o software especificado. Entretanto, por ser uma forma não exaustiva de validação, sua utilização é mais indicada para o acompanhamento da execução do sistema, permitindo verificar os caminhos de execução julgados críticos.

Foram elaborados vários casos de execução do AIDA para o sistema concorrente, com a utilização do sistema por mais de um usuário ao mesmo tempo, executando cada um deles várias ferramentas, para testar, entre outras coisas, o tratamento de erros pelo sistema. Parte de um exemplo de simulação é apresentado na Figura 14.

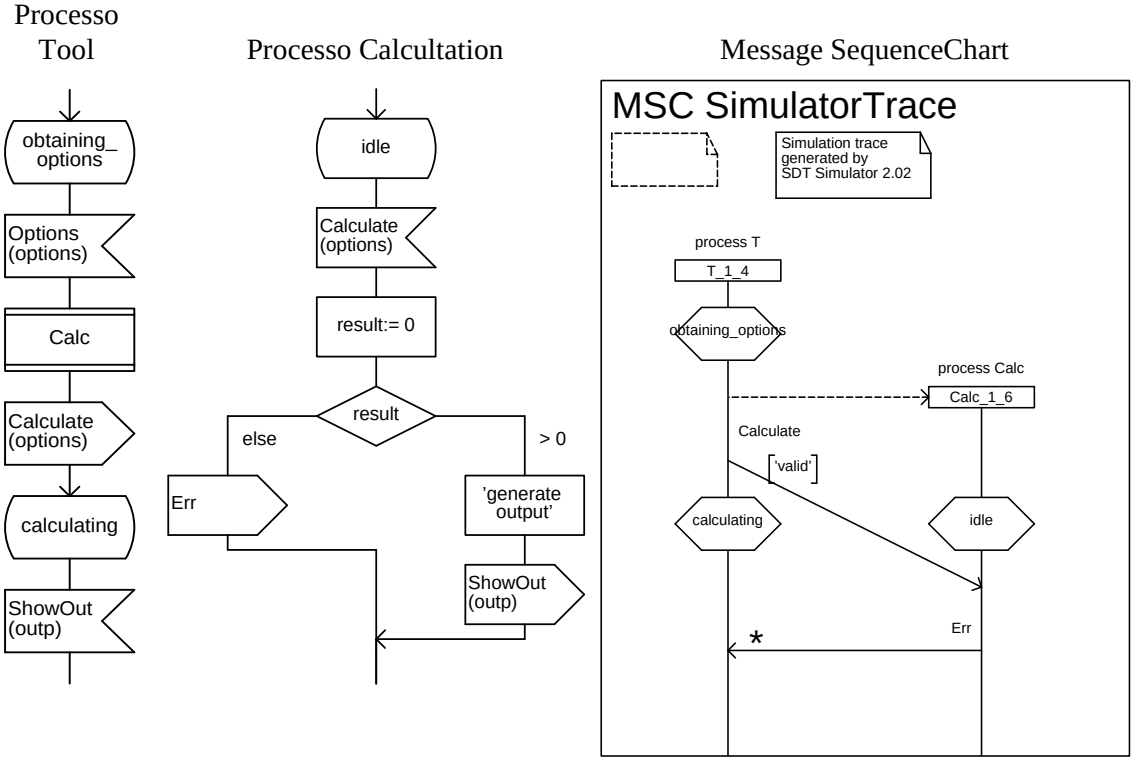

Figura 14 - Exemplo de simulação

O primeiro diagrama representa parte do processo *Tool* e o segundo parte do processo *Calculation*. O processo *Tool* encontra-se no estado *obtaning_options*, recebe as opções de execução do processo *TInterf*, cria uma instância *Calc* do processo *Calculation*, envia para ela o sinal *Calculate* , com as opções de execução, e vai para o estado *calculating*, esperando pela resposta do cálculo do algoritmo. O processo *Calculation* quando é criado vai para o estado *idle*, recebe o sinal *Calculate* do processo *Tool*, executa os cálculos, que para simular a ocorrência de um erro, teve zero atribuído ao seu resultado. O sinal de erro *Err* é então enviado para o processo *Tool*. Entretanto, *Tool* se encontra no estado *calculating*, que está preparado para receber apenas o sinal *ShowOut*, que indica a exibição, para o usuário, do resultado do cálculo do algoritmo. Assim, uma falha na especificação é detectada e a simulação do sistema é interrompida. Como na validação, com a identificação de falhas na especificação, retorna-se à especificação em SDL e eventualmente ao seu modelo OMT, para as devidas alterações.

O terceiro diagrama representa parte de um MSC - Message Sequence Chart[6], que consiste num tipo de digrama de eventos e serve para representar dinamicamente a simulação do sistema. Na Figura 14 apenas as instâncias dos processos *Tool* e *Calculation*, e os sinais trocados por elas estão representados. O asterisco indica que o sinal *Err* não foi consumido pelo processo *Tool* e a simulação foi interrompida.

8. Conclusão

O uso da Object Modeling Technique - OMT tem se mostrado como uma excelente opção no processo de desenvolvimento de software. Dentre outras coisas, permite o reuso de software, a abstração de detalhes e provê modularidade aos sistemas, tornando as atividades de manutenção e evolução menos complexas. Entretanto, esta técnica não consiste numa notação tão rigorosa, permitindo interpretações ambíguas dos modelos elaborados, além de não possibilitar a simulação, a validação ou a geração de código para prototipação do sistema em desenvolvimento.

Assim, o uso combinado de OMT com SDL visa impor um certo formalismo ao processo de desenvolvimento, aparecendo como uma metodologia bastante interessante para a especificação e o desenho de sistemas que utilizam a orientação a objetos, e que são passíveis de serem evoluídos. Dentre outras coisas a linguagem SDL permite a representação da estrutura do sistema, bem como do seu comportamento, em diferentes níveis de abstração, além de tornar possível a validação e prototipação rápida do sistema especificado.

Este trabalho apresentou o uso desta metodologia na evolução do sistema centralizado para uma versão concorrente do AIDA - Ambiente Integrado para Desenvolvimento e Análise, um ambiente para o gerenciamento e a análise de dados experimentais gerados na Empresa Brasileira de Pesquisa Agropecuária - EMBRAPA. Além da validação e da simulação do sistema, um dos principais resultados obtidos, foi a facilidade que esta metodologia oferece para a evolução de sistemas através da utilização de conceitos como herança e reuso, presentes na OMT, em conjunto com construções de SDL como *package*, *virtual* e *redefined block* e *process types.*

9. Referências Bibliográficas

[1] BELINA, F.; HOGREFE, D.; SARMA, A. *SDL with applications from protocol specification*. Prentice Hall International, 1991. 275p.

[2] CHAIM, M.L.; TERNES, S.; DELFINO A.; Aoki, R.; ALVIM, L.; MEDEIROS, S.; MACÁRIO, C.G.N.; MOURA, M.F.; HIGA, R.H.; ARANTES, M.P.C.; PORTO, J.R.; BACARIN, E.; FESTA, M. *Ambiente Integrado para Desenvolvimento e Análise - AIDA*. Campinas: EMBRAPA-CNPTIA, 1995 não paginado (EMBRAPA-CNPTIA. Projeto 12.0.96.121.00).

[3] EMBRAPA/ Centro Nacional de Pesquisa Tecnológica em Informática para a Agricultura(Campinas, SP). *Software AIDA - especificação de requisitos.* Campinas, 1997.

[4] FÆRGEMAND O.; OLSEN A. New features in SDL-92. Disponível*: site SDL[literature*]. URL:http://www.dr.dk/public/SDL/litt.html#papers Consultado em 20 fev. 1997.

[5] ITU-T(Geneve, Switzerland). *Recommendation Z.100 CCITT specification and description language (SDL): programming languages.* Geneve, 1993.

[6] ITU-T(Geneve, Switzerland). *Recommendation Z.120. message sequence chart* (MSC). Geneve, 1993.

[7] MACÁRIO, C.G.N.; BONFIM, W.S.; CHAIM, M.L.; ANTUNES, J.F.G.; TERNES, S.; AOKI, R.; ALVIM, L.; PACHECO, O.I.P.; PALMIERI, S.; FESTA, M.N.; GASPAR, D.M.; SERRA, R.; HIGA, R.H.; ARANTES, M.P.C. *Evolução do Ambiente de Software NTIA*. Campinas: EMBRAPA-CNPTIA, 1994. não paginado. (EMBRAPA-CNPTIA. Projeto 12.0.94.071.00).

[8] MACÁRIO, C.G. N. *O uso combinado da técnica de modelagem baseada em objetos OMT com a linguagem de especificação formal SDL como metodologia alternativa para o desenvolvimento do ambiente de software AIDA*. Campinas: Universidade Estadual de Campinas - Faculdade de Engenharia Elétrica e de Computação - Departamento de Telemática, Tese de Mestrado, dezembro/1997. 144p.

[9] MACÁRIO, C.G. N.; PEDROSO JÚNIOR, M.; BORELLI, W. C. OMT+SDL: An alternative methodology for the AIDA development. *In: III CONGRESO INTERNACIONAL DE INGENIERIA INFORMATICA*, Buenos Aires. ICIE 96-97: proceedings. Buenos Aires:Universidad de Buenos Aires - Faculdad de Ingenieria - Departamento de Computacion, abril/1997. p.438-448.

[10] MACÁRIO, C.G. N.; PEDROSO M. J; BORELLI, W. C. Designing a multi-user software environment for development and analysis using a combination of OMT and SDL92. *In:SDL'97 TIME FOR TESTING SDL, MSC AND TRENDS, 8th*., 1997, Evry-France. Eighth SDL Forum: 1997: proceedings. Amsterdam: Elsevier Science Publishers, setembro/1997. p.03-17.

[11] RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; Eddy,F.; Lorensen, W. *Object oriented modeling and design*. New Jersey: Prentice Hall International, 1991. 500p.

[12] TELELOGIC AB (Malmö, Sweden). *Getting started with SDT3.02*. Malmö, 1995.